

Software Decoys for Software Counterintelligence

by Dr. Neil C. Rowe, Dr. J. Bret Michael, Dr. Mikhail Auguston, and Mr. Richard Riehle

Some information systems are critical to defend against malicious attack. Yet they often rely on just the same countermeasures as any system—firewalls, authentication, intrusion detection systems, and encryption—although politically motivated attackers may be far more determined than hackers to bring them down. Future information security will increasingly use ideas from military defensive tactics³ to effectively defend critical information systems. This will include automatic “counterintelligence” with deliberately deceptive behavior, what we call “software decoys.” Decoys can deceive attackers into thinking their attacks have succeeded while protecting key assets at least temporarily.

Much good work has been done on intrusion detection systems,^{8, 11} but only recently has there been corresponding work on how an attacked system should respond. Many respond to serious attacks by turning off the network connection, a high cost in today's networked world. Such a response tells the attacker they have been detected and this may just direct them to better targets. Moreover, defenders lose information about how the attack would have proceeded which they could have used to make defense more effective.

A cyber-attack is an attack on resources to gain tactical or

strategic advantage just as in regular warfare. Deceptive responses can be automated much like the attacks. Such responses can be very effective because attackers depend on the honesty of the computer systems they attack. Deception can confuse their planning or frustrate them for a while without giving away our recognition that we are being attacked. This could be especially important during intensive information warfare when terrorists attempt to bring down critical systems in a short period of time: Delay permits time to analyze the attack and plan a response. Deception also allows us to turn an attacker's own strengths of patience and determination against them, much as Asian martial arts like Akido do with physical attacks.

The Concept of a Software Decoy

We have been researching “intelligent software decoys”⁹. We use this term to cover a spectrum of deceptive defensive activity¹. This can range from mimicking normal behavior of the computer system (as when an attacker thinks they have gained system administrator privileges and we pretend they can modify key directories), through inventing appealing activities for the attacker (as when an attacker overflows a buffer and we pretend they have changed the behavior of

the operating system), to new facilities (as when an attacker gets clues to a trap site with apparently vulnerable software). Appropriate deceptive tactics depend on the value of the resources being protected and the danger of the attack. But the general idea is to limit or confine⁷ attacks that get through our first line of defense rather than stop attacks. Decoys differ from honeypots⁴ in providing defense, not data.

Decoys are easiest to make when simple effects (like denial-of-service) are sought by attackers. They will generally work best against hands-on adversaries as opposed to automated scripts, though unpredictable responses by a decoy could well foil a script. Effective decoys need not be complex. Simple ploys in warfare can be surprisingly effective when their timing is right, they are consistent with enemy expectations, and they have some creativity.

Decoying capabilities should be distributed through an operating system and applications programs to provide a uniform front to attackers with no single point of compromise. They could go in Web servers, mail servers, and file-transfer utilities to address denial-of-service attacks and attempts to jump into the operating system. They could go in directory-listing capabilities to provide false information about sensitive di-

rectories. They could also go in network routers to address denial of service and suspicious patterns (like strings of nulls) with connection errors. More ambitious decoys could be embedded in all file-writing capabilities or all security-related activities of the operating system, through the use of “wrapper” technology that automatically inserts checking code around sensitive statements (“instruments” it).⁹ While this may sound ambitious, an analogous technology exists for instrumenting code to calculate software metrics and monitor software at runtime, and such instrumentation has been successfully accomplished for large software systems—it is not hard for simple open-source operating systems like those for small devices.

We can distinguish levels of decoying. At the simplest level are memoryless decoys that respond the same way to the same local context. A behavior model based on an “event grammar” can operate on the system log to detect suspicious local context. It can use sophisticated ideas from the field of temporal logic. Creativity of decoy responses can be done with generative grammars having random choices. For instance, we have written generators for fake error messages (like “Error at 2849271: Segmentation fault”) and for fake directory listings (with fake file names, dates, sizes, and subdirectories).

At one level, a decoy can remember other invocations of the same code. For instance, a server can store details of other transactions it has serviced so that it can recognize denial-of-service attacks. At another level, decoys in different soft-

ware modules can share information about an attack, as when an attacker installs their own operating system. Finally at the highest level, a decoy can simulate the entire operating system itself within a “sandbox” or safe environment. This would be helpful when Trojan horses of unknown capabilities have been inserted into an operating system and the decoy must simulate them.² The higher levels of decoying require an architecture of response management.⁶

Types of Software Decoy Responses

A generally useful decoy tactic is the exaggeration of intended attacker effects: *Good deceptions should confirm preconceptions of the deceived.* Under a denial-of-service attack, for instance, we can pretend to increase the load on a computer system by deliberately delaying system responses. This can be done by calculating and implementing delays, accomplished by additional process-suspension time, into the servicing of attacker transactions,¹⁰ with perhaps additional scripted interaction with the attacker.

An important factor in this is the probability that we are under attack. Unfortunately, new vulnerabilities and new techniques for exploiting those vulnerabilities are constantly being discovered. Recent hacker behavior shows an increasing automation of attacks, increasing use of rootkits, decreasing use of probes, and an increasing use of encryption for network communication.⁴ But a determined adversary like a terrorist group will want to try new methods we have

not anticipated. We must then use general principles to estimate the probability we are under attack, and respond proportionately to this probability. We can use current intrusion detection methods for this, both anomaly and misuse detection, but an especially helpful clue we are investigating are reports from similar sites about attacks that they are undergoing.⁵ Automatic data mining from system logs can be helpful at those sites to analyze how they were attacked.

Decoy delays can be accomplished by process-suspension time alone, but alternatives can make the deception more interesting and engaging to the attacker. Many attackers see their activities as like playing a computer game, so some game-like behavior in the decoy could be helpful, as could “showmanship.”¹¹ This could involve user interactions such as requests for authorization, requests to confirm allocation of more system resources like memory, deliberate errors, and invocation of new scripts pretending to be system-administrator tools—we can get creative.

Responses of software decoys must necessarily vary with resources available to fight the attack. Consider denial-of-service decoys for transaction servers. Delay exaggeration is only effective below a certain system load, because good deception requires that we still process the transactions albeit more slowly. If the attack intensity continues to increase, we could systematically simplify transactions, without telling users, by ignoring less important parts of the input. Or we could respond to a transaction with a cached result of a similar transaction, an effective idea for



denial-of-service attacks doing the same transaction repeatedly.

If the attack intensity continues to grow, the system has no choice but to refuse transactions. However, we may still fool an attacker if we substitute a low-resource interaction that could conceivably result from a successful attack. For instance, we could say "Buffer overflow" and start what appears to be a debugger with "Stopped at line 368802 of module serv89—singlestep?" Or we could claim memory needs to be reallocated due to the high system load, and give the attacker a fake opportunity to change module memory requirements. Eventually however, if attack intensity continues to increase we must turn off the network connection and terminate the game with the attacker.

Attackers will eventually recognize decoys, and will plot countermeasures such as ignoring sites with recognizable decoy "signatures." But we can plot to counter the countermeasures, and so on. The classic field of game theory provides methods to analyze such situations and find our best overall strategy.

Endnotes

1. Bell, J. B., & Whaley, B., *Cheating and Deception*, New Brunswick, NJ, St. Martin's Press, 1991.
2. Bressoud, T. & Schneider, F. B., "Hypervisor-based Fault-tolerance. ACM Transactions on Computer Systems," Vol. 14, No. 1, pp. 80–107, Feb. 1996.
3. Fowler, C. A., & Nesbit, R. F., "Tactical deception in air-land warfare," *Journal of Electronic Defense*, Vol. 18, No. 6, pp. 37–44 & 76–79, June 1995).
4. The Honeynet Project, *Know Your Enemy*, Boston, Addison-Wesley, 2002.
5. Ingram, D., Kremer, H., & Rowe, N., "Distributed Intrusion Detection for Computer Systems

Using Communicating Agents," *Proceedings from the 6th International Symposium on Research and Technology on Command and Control*, Annapolis, MD, June 2001.

6. Lewandowski, S., Van Hook, D., O'Leary, G., Haines, J., & Rossey, L., SARA: "Survivable Autonomic Response Architecture," *Proceedings from DARPA Information Survivability Conference*, Anaheim CA, June 2001, Vol. 1, pp. 77–88.
7. Liu, P. & Jajodia, S., "Multi-phase Damage Confinement in Database Systems for Intrusion Tolerance," *Proceedings from the 14th Computer Security Foundations Workshop*, Cape Breton, NS, pp. 191–205, June 2001.
8. Lunt, T. F., "A Survey of Intrusion Detection Techniques," *Computer and Security*, Vol. 12, No. 4, pp. 405–418, June 1993.
9. Michael, B., Auguston, M., Rowe, N., & Riehle, R., "Software Decoys: Intrusion Detection and Countermeasures," *Proceedings from the 2002 Workshop on Information Assurance*, West Point, NY, June 2002.
10. Somayaji, A., & Forrest, S., "Automated Response Using System-call Delays," *Proceedings from the 9th USENIX Security Symposium*, pp. 185–197, August 2000.
11. Vigna, G. & Kemmerer, R. A., "NetSTAT: A Network-based Intrusion Detection Approach," *Proceedings from the 14th Annual Computer Security Applications Conference*, Scottsdale, AZ, pp. 25–34, December 1998.

Biographies

Dr. Neil C. Rowe is Professor and Associate Chair of Computer Science at the U.S. Naval Postgraduate School where he has been since 1983. He has a Ph.D. in Computer Science from Stanford University (1983), and E.E. (1978), S.M. (1978), and S.B. (1975) degrees from the Massachusetts Institute of Technology. He has done research on intelligent access to multimedia databases, information security, image processing, robotic path planning, and intelligent tutoring systems. He has authored over one hun-

dred technical publications and a book. He may be reached at ncrowe@nps.navy.mil.

Dr. James Bret Michael has been Associate Professor of Computer Science at the U.S. Naval Postgraduate School, Monterey California since 1998. He received his M.S. (1987) and Ph.D. (1993) degrees from the School of Information Technology and Engineering at George Mason University, and B.S. (1983) from West Virginia University. His research interests include both information operations and computer security for distributed computing systems such as those in missile defense, has authored over fifty technical publications and a book, and is a senior member of the IEEE. He may be reached at bmichael@nps.navy.mil.

Dr. Mikhail Auguston is an Associate Professor of Computer Science at New Mexico State University. He has graduated summa cum laude in Mathematics from University of Latvia in 1971 and received Ph.D. in Computer Science from the Glushkov Institute of Cybernetics in Kiev (USSR) in 1983. He has more than thirty years of research experience in programming language design and implementation, program testing, and debugging tool design, and has authored more than sixty technical publications. He may be reached at mikau@cs.nmsu.edu.

Mr. Richard Riehle is a Visiting Professor of Computer Science at U.S. Naval Postgraduate School. He is also a Principal of AdaWorks Software Engineering, a consulting firm that specializes in software development and training in Ada. Mr. Riehle has over twenty-five years in software development in both military and non-military systems. His current interests are software architecture, software deception techniques, programming language design, and software reliability. He has a B.S. from Brigham Young University and an M.S. in Software Engineering from National University. He may be reached at rdriehle@nps.navy.mil.